Symbolically Synthesizing Small Circuits

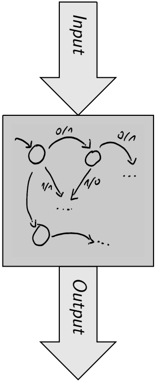
Rüdiger Ehlers[1] Robert Könighofer[2] Georg Hofferek[2]

[1]University of California at Berkeley
United States

[2]Graz University of Technology
Austria

FMCAD 2012 – 24^{th} October 2012

Verification:



$+$

$\mathbf{G}(u = 0 \rightarrow \mathbf{X}(v = 1))$

$\Rightarrow$

Satisfied

Not satisfied
(+ counter-example)

Synthesis:

$\mathbf{G}(u = 0 \rightarrow$
$\mathbf{X}(v = 1))$

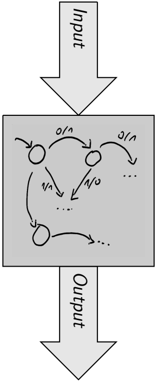
$+$

Input $= \{u, \ldots\}$
Output $= \{v, \ldots\}$

$\Rightarrow$

Realisable $\longrightarrow$

Not realisable



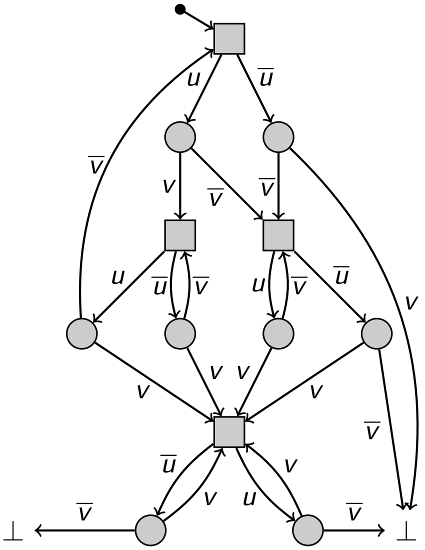
Synthesis in a nutshell

$$\mathbf{G}(u = 0 \\ \rightarrow \ \mathbf{X}(v = 1))$$

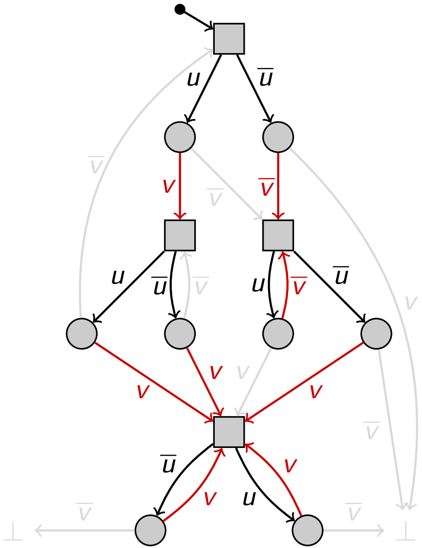
$$+$$

$$\text{Input} = \{u, \ldots\}$$
$$\text{Output} = \{v, \ldots\}$$

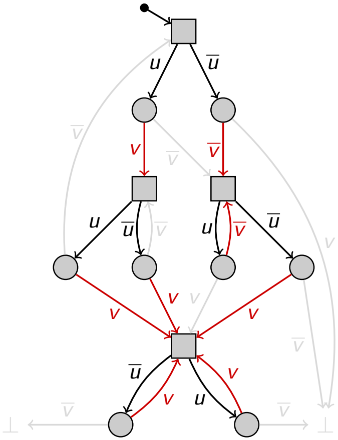
Synthesis in a nutshell



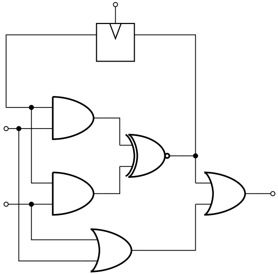
Synthesis in a nutshell



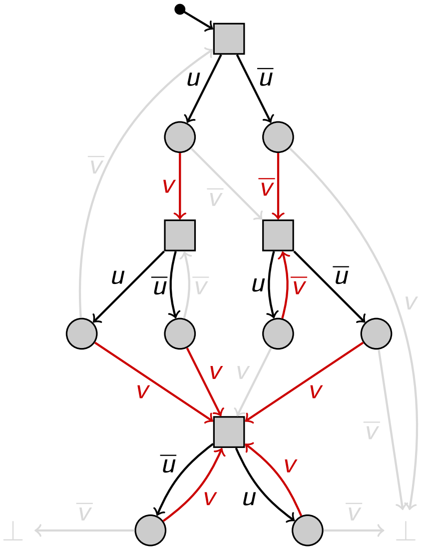
Synthesis in a nutshell



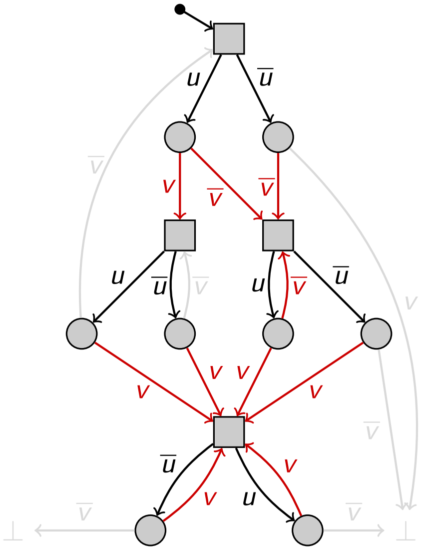

```
int main() {
  bool s = 1;
  while (true) {
    bool b = in-u();
    out-v(b | s);
    s = s ⊕ u;
    wait();
  }
}
```



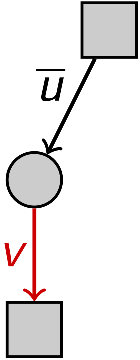
On general strategies



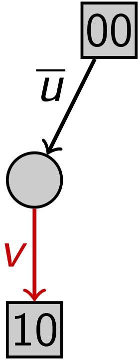
On general strategies



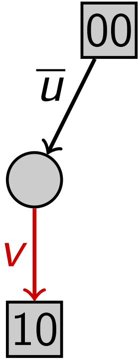
Encoding general strategies



Encoding general strategies



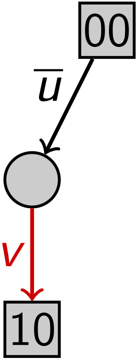
Encoding general strategies



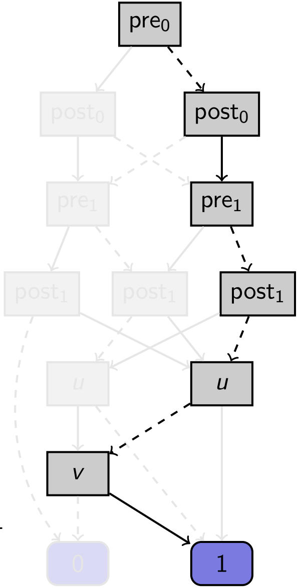
$$\Downarrow$$

$$\overline{pre_0} \wedge \overline{pre_1} \wedge \overline{u} \wedge v \wedge pre_0 \wedge \overline{pre_1}$$

Encoding general strategies



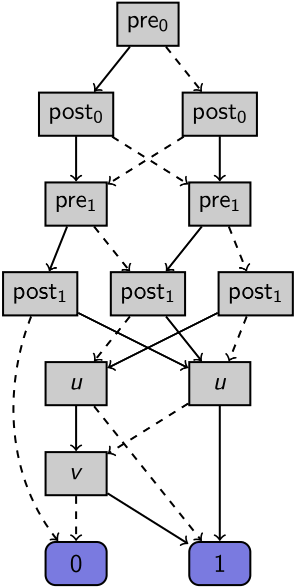
$\Rightarrow$



$\Downarrow$

$$\overline{pre_0} \wedge \overline{pre_1} \wedge \overline{u} \wedge v \wedge pre_0 \wedge \overline{pre_1}$$

Building circuits from general strategies



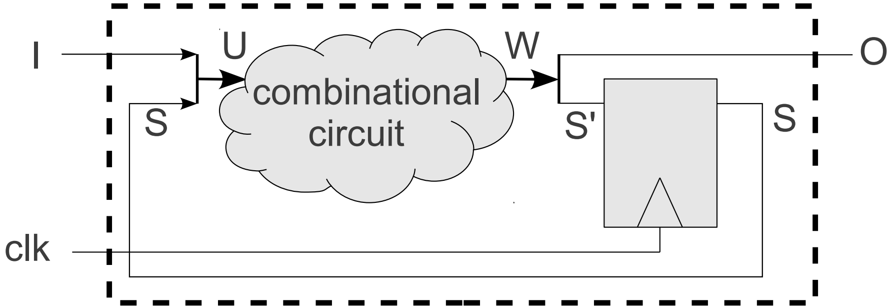
General strategies ...

- ... represent allowed behaviour of the system to synthesize
- ... are typically given symbolically
- ... are often **huge**. To have 64 state bits in the BDD is common.

What is a suitable implementation?

Any implementation that is a specialisation of the general strategy is fine.

Building circuits from general strategies

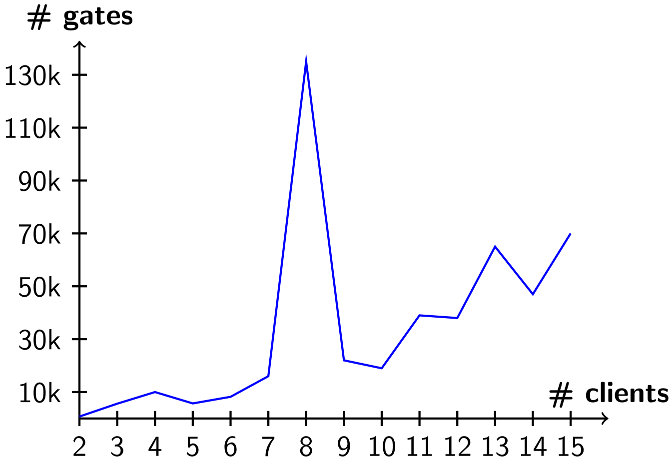


Previous approaches to building the combinational circuit

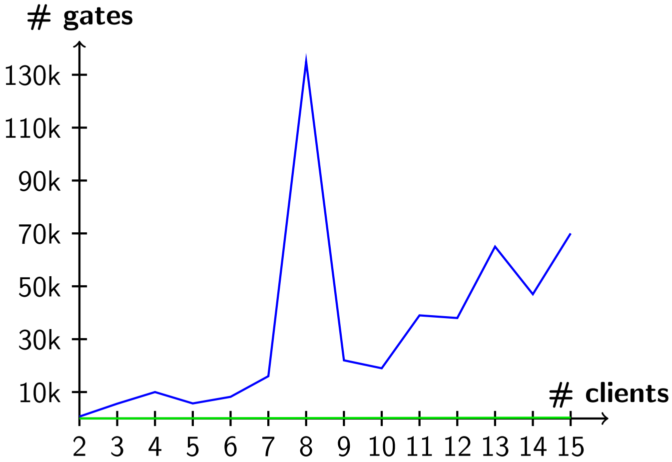
Experiences in the synthesis domain

- Kukula and Shiple (2000) - huge, deep circuits
- Bañeres et al. (2004) - extremely slow on synthesis problems
- Bloem et al. (2007b) - best approach so far, circuits are still large
- Jiang et al. (2009) - generates larger circuits than the approach by Bloem et al. (2007b)
- ...

Example: AMBA AHB bus arbiter (Bloem et al., 2007b)



Example: AMBA AHB bus arbiter (Bloem et al., 2007b)



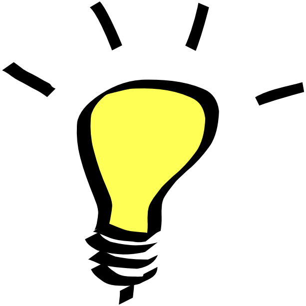
Our new approach

Main idea

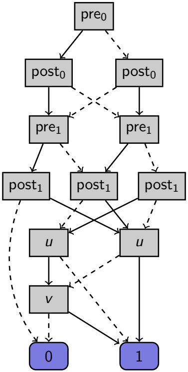
Use **computational learning of Boolean functions** as main tool for computing small circuit implementations.

Benefits

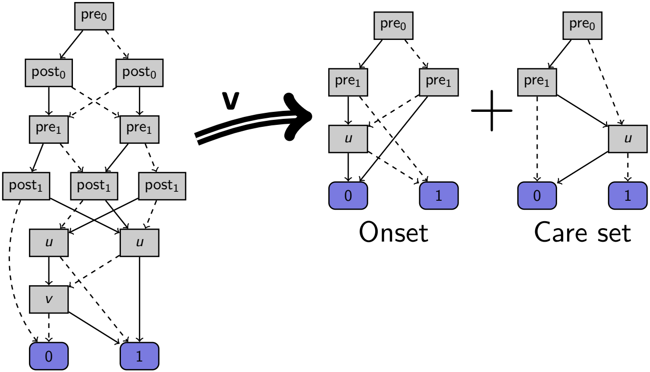
- We can efficiently benefit from a large number of *don't care* cases
- We obtain *shallow* circuits



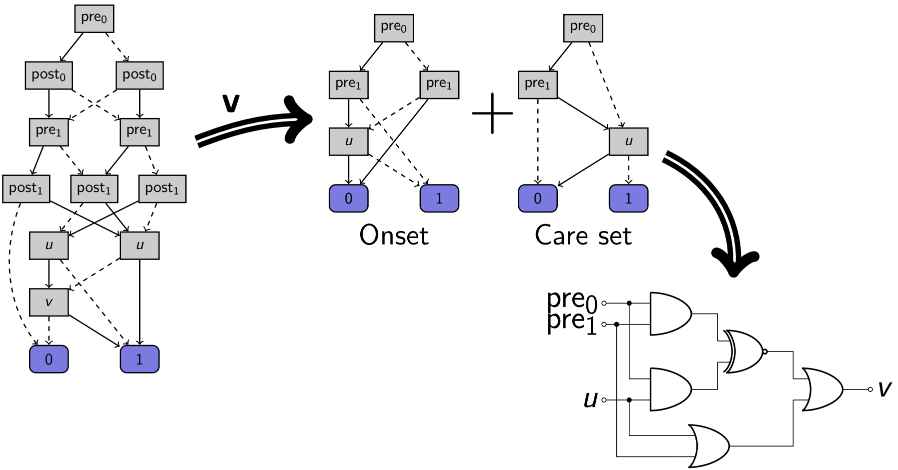
Bit-by-bit circuit extraction approach



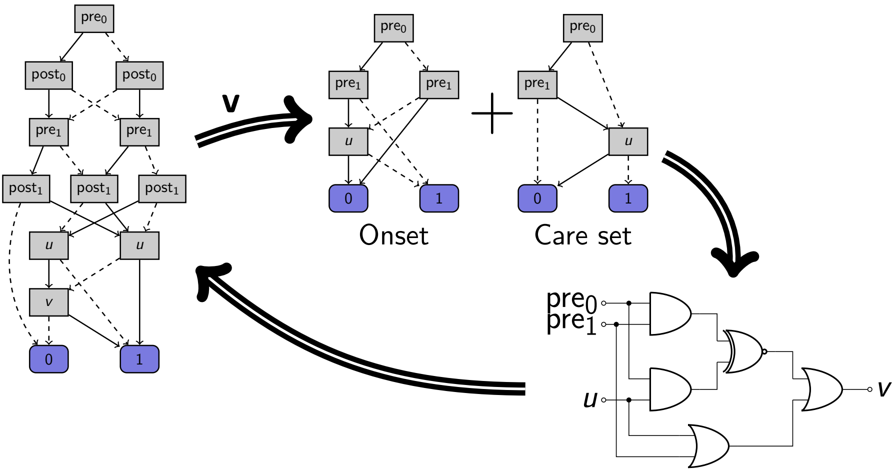
Bit-by-bit circuit extraction approach



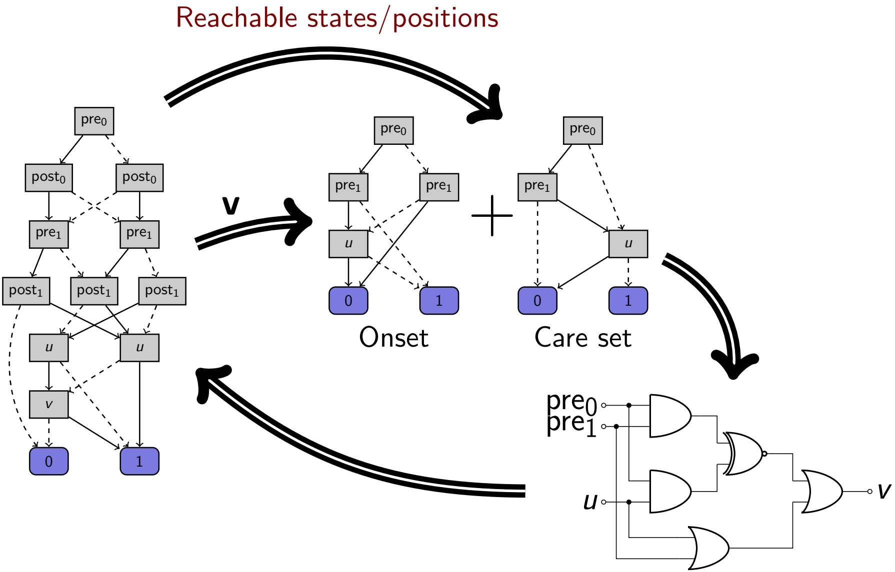
Bit-by-bit circuit extraction approach



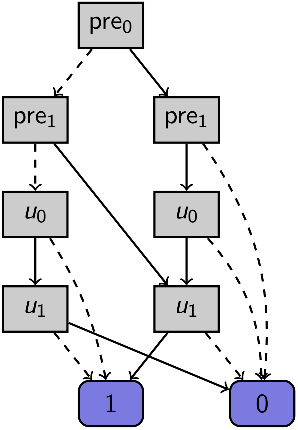
Bit-by-bit circuit extraction approach



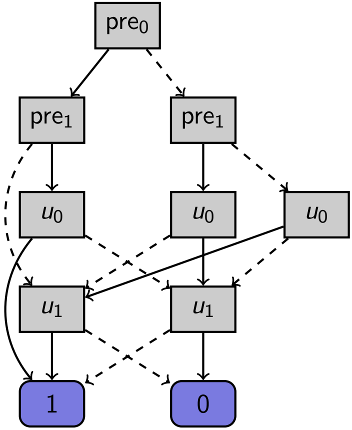
Bit-by-bit circuit extraction approach



Boolean formula learning (DNF)

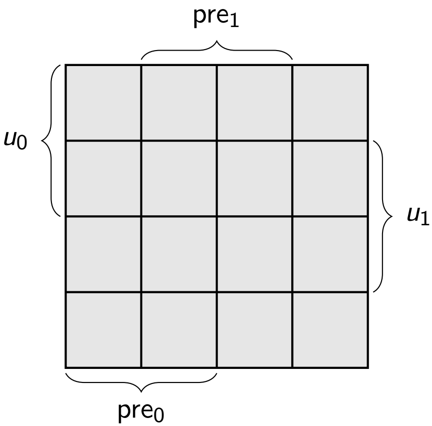


Onset



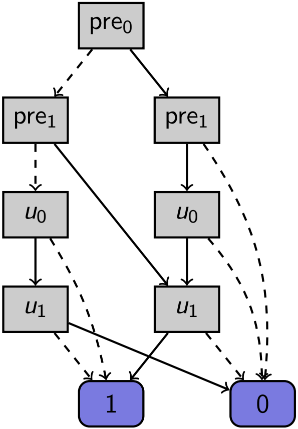
Care Set

Boolean formula learning (DNF)



Boolean formula learning (DNF)

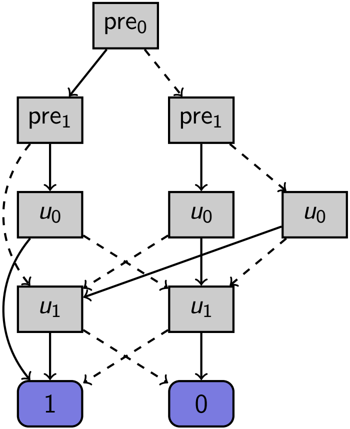
		pre_1	pre_1		
u_0	0	0	0	1	
u_0	0	1	1	0	u_1
	0	0	1	1	u_1
	0	0	0	1	
	pre_0	pre_0			



Onset

Boolean formula learning (DNF)

		pre_1	pre_1	
u_0	X	0	0	X
u_0, u_1	0	1	X	0
u_1	0	X	1	X
	X	0	X	1
	pre_0	pre_0		



Care Set

Boolean formula learning (DNF)

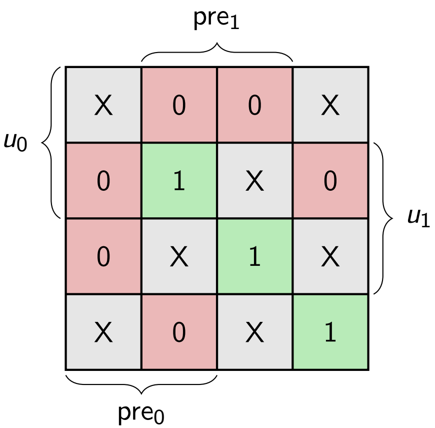
pre_1

u_0	X	0	0	X
u_0	0	1	X	0
u_1	0	X	1	X
	X	0	X	1

pre_0

u_1

Boolean formula learning (DNF)



Candidate solution:

Boolean formula learning (DNF)

		pre_1			
	X	0	0	X	
u_0	0	1	X	0	
	0	X	1	X	u_1
	X	0	X	1	
	pre_0				

Candidate solution:

$\psi =$ **false**

Boolean formula learning (DNF)

pre_1

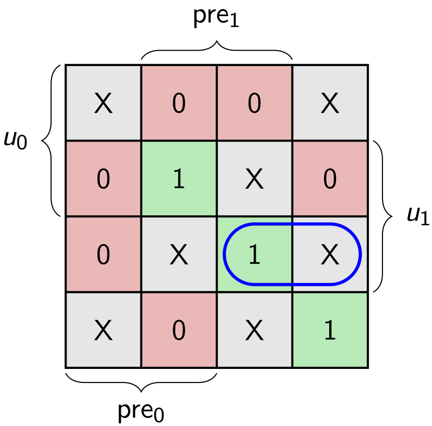
u_0	X	0	0	X
	0	1	X	0
u_1	0	X	1	X
	X	0	X	1

pre_0

Candidate solution:

$\psi =$ **false**

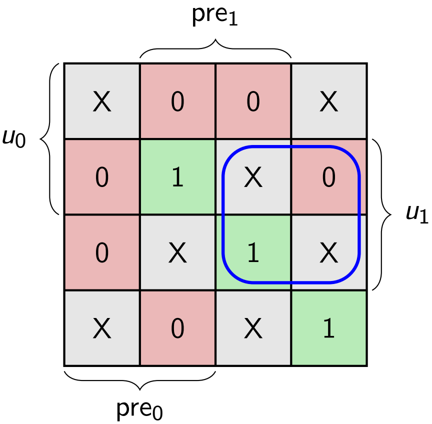
Boolean formula learning (DNF)



Candidate solution:

$\psi =$ **false**

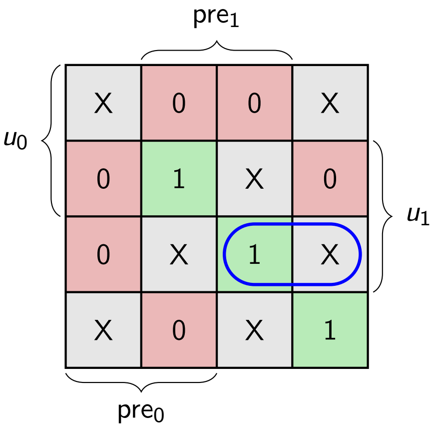
Boolean formula learning (DNF)



Candidate solution:

$\psi =$ **false**

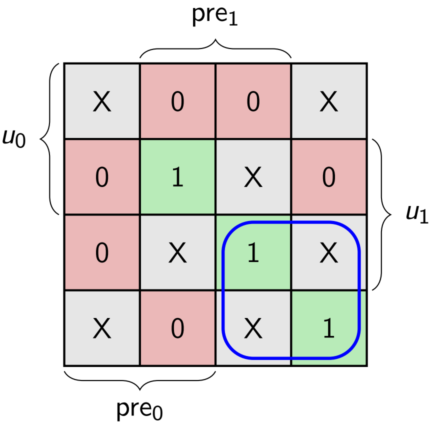
Boolean formula learning (DNF)



Candidate solution:

$\psi =$ **false**

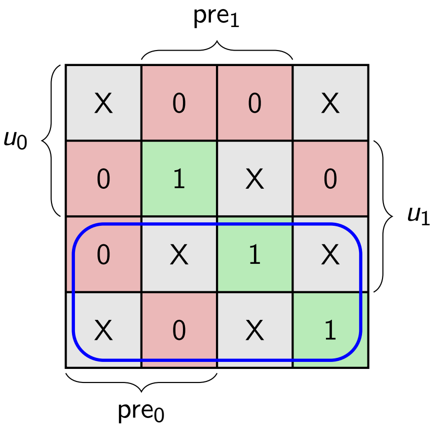
Boolean formula learning (DNF)



Candidate solution:

$\psi =$ **false**

Boolean formula learning (DNF)



Candidate solution:

$\psi =$ **false**

Boolean formula learning (DNF)

$\overbrace{\qquad\qquad}^{\mathsf{pre}_1}$

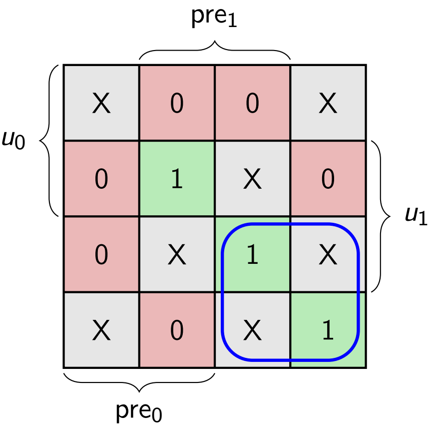
u_0	X	0	0	X
u_0, u_1	0	1	X	0
u_1	0	X	1	X
	X	0	X	1

$\underbrace{\qquad\qquad}_{\mathsf{pre}_0}$

Candidate solution:

$\psi =$ **false**

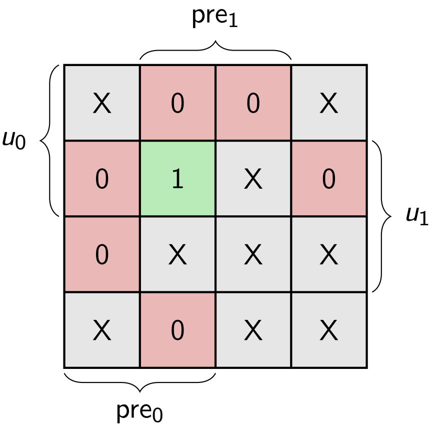
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg \mathsf{pre}_0 \wedge \neg u_0$

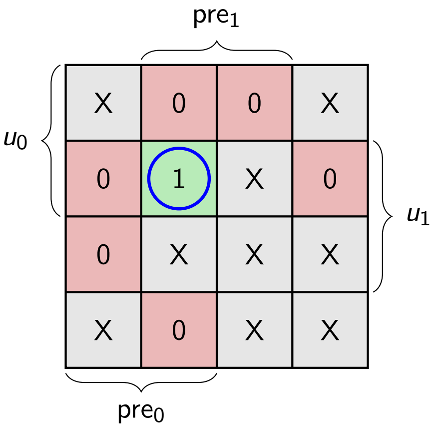
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg pre_0 \wedge \neg u_0$

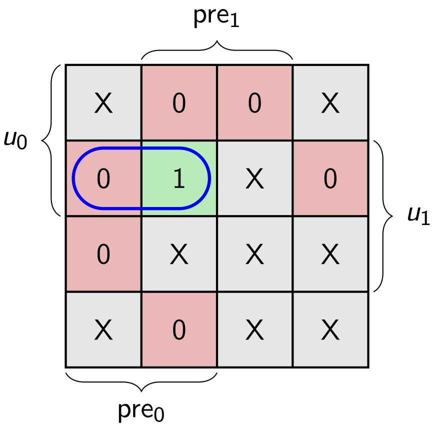
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg \mathsf{pre}_0 \wedge \neg u_0$

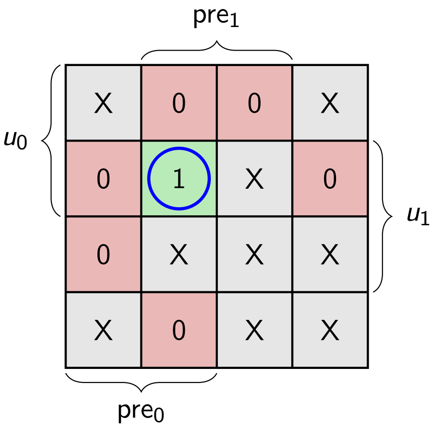
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg \mathsf{pre}_0 \wedge \neg u_0$

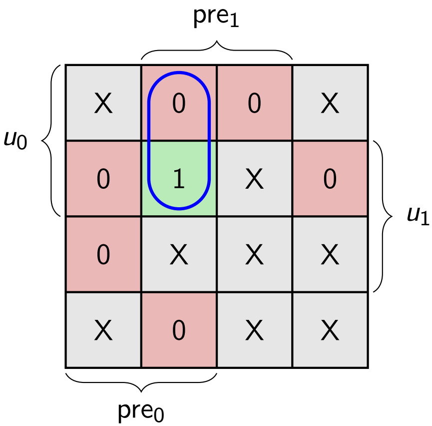
Boolean formula learning (DNF)



Candidate solution:

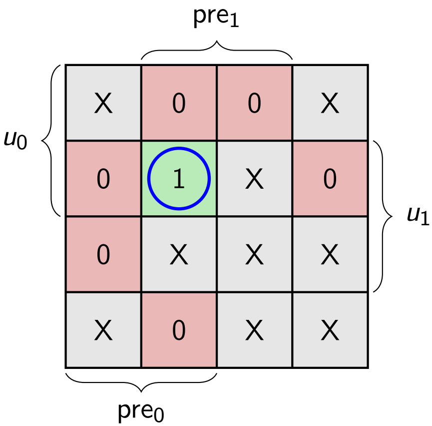
$\psi = \neg \mathrm{pre}_0 \wedge \neg u_0$

Boolean formula learning (DNF)



Candidate solution:

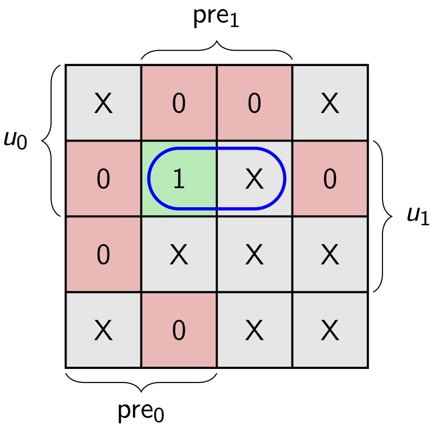
$\psi = \neg \mathrm{pre}_0 \wedge \neg u_0$



Candidate solution:

$\psi = \neg pre_0 \wedge \neg u_0$

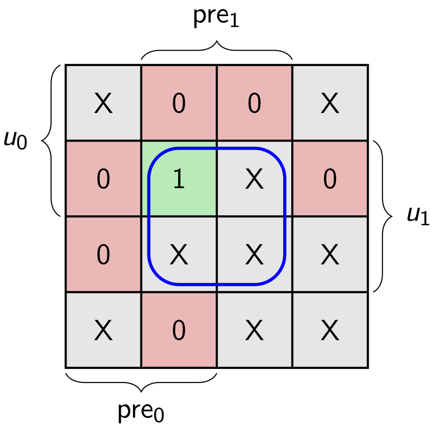
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg \mathsf{pre}_0 \wedge \neg u_0$

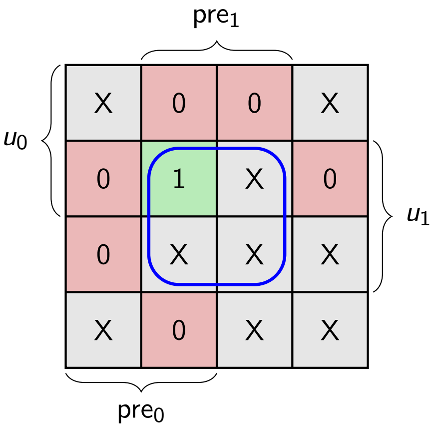
Boolean formula learning (DNF)



Candidate solution:

$\psi = \neg \mathrm{pre}_0 \wedge \neg u_0$

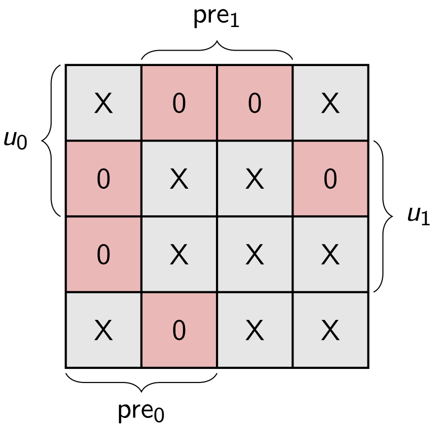
Boolean formula learning (DNF)



Candidate solution:

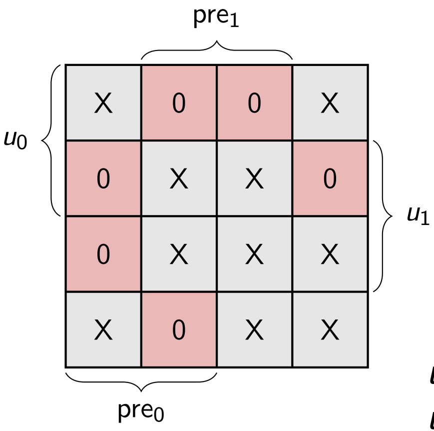
$\psi = (\neg \mathsf{pre}_0 \wedge \neg u_0) \vee (\mathsf{pre}_1 \wedge u_1)$

Boolean formula learning (DNF)



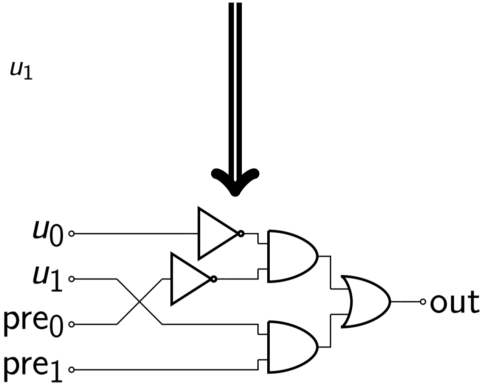
Candidate solution:

$\psi = (\neg \mathsf{pre}_0 \land \neg u_0) \lor (\mathsf{pre}_1 \land u_1)$

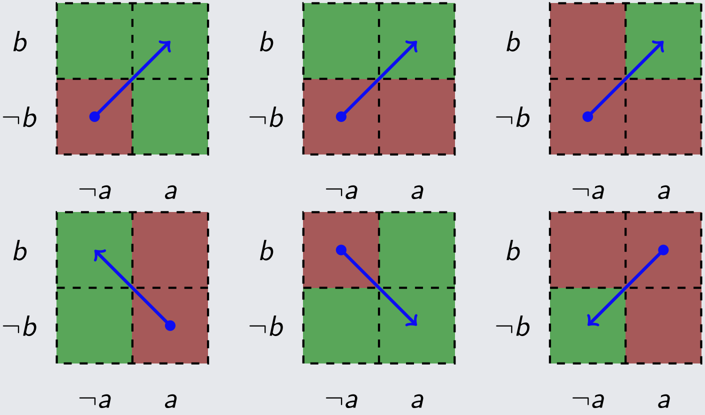


Candidate solution:

$\psi = (\neg \mathrm{pre}_0 \wedge \neg u_0) \vee (\mathrm{pre}_1 \wedge u_1)$

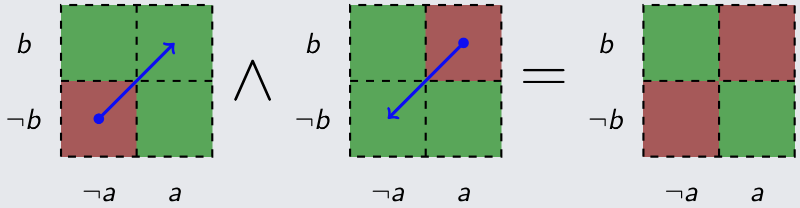


Z-monotone functions



Overlay example: decomposing a XOR into CDNF

Idea



$(a \vee b) \quad \wedge \quad (\neg a \vee \neg b) \quad =$ Target function

CDNF learning

Algorithm

While target function not found:

- Search for **false-positive** z (as in CNF)
  If found, add a new z-based DNF
- Search for **false-negative** z (as in DNF)
  If found, update all DNFs to accept z

Example: learning a XOR b

Candidate solution

(**true**)

Operation

Check for false-positive

Example: learning a XOR b

Candidate solution

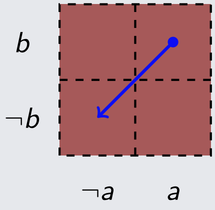
(**true**)

Operation

Check for false-positive $\rightarrow$ ✘: ab

Example: learning a XOR b

Candidate solution



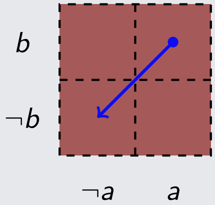
(**false**)

Operation

Check for false-negative

Example: learning a XOR b

Candidate solution



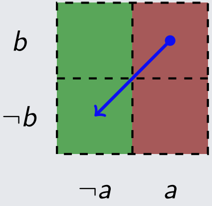
(**false**)

Operation

Check for false-negative $\rightarrow$ ✘: $\bar{a}b$

Example: learning a XOR b

Candidate solution



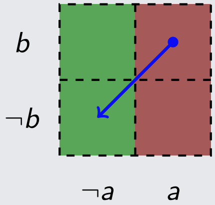
$(\neg a)$

Operation

Check for false-positive

Example: learning a XOR b

Candidate solution



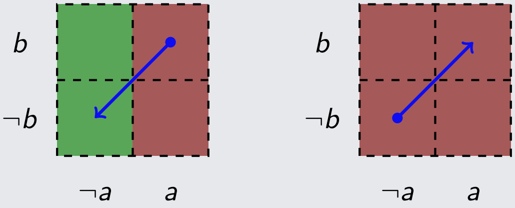
$(\neg a)$

Operation

Check for false-positive $\rightarrow$ ✘: $\overline{ab}$

Example: learning a XOR b

Candidate solution



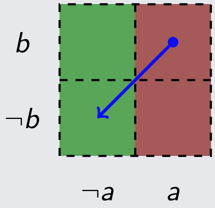
$$(\neg a) \quad \wedge \quad (\textbf{false})$$

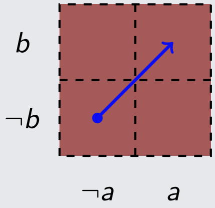
Operation

Check for false-negative

Example: learning a XOR b

Candidate solution





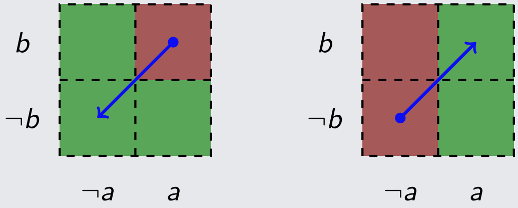
$(\neg a) \quad \wedge \quad (\textbf{false})$

Operation

Check for false-negative $\rightarrow$ ✘: $\overline{b}a$

Example: learning a XOR b

Candidate solution



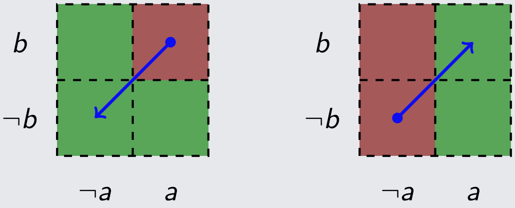
$(\neg a \vee \neg b) \quad \wedge \quad (a)$

Operation

Check for false-positive

Example: learning a XOR b

Candidate solution



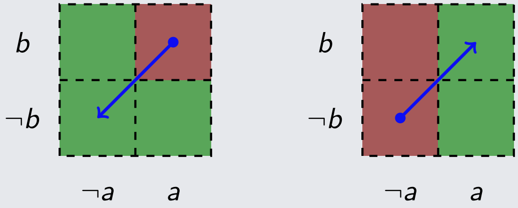
$$(\neg a \vee \neg b) \quad \wedge \quad (a)$$

Operation

Check for false-positive $\rightarrow$ ✔

Example: learning a XOR b

Candidate solution



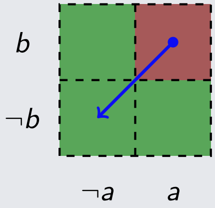
$$(\neg a \vee \neg b) \quad \wedge \quad (a)$$

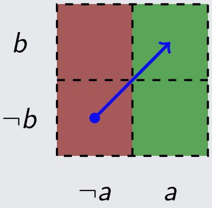
Operation

Check for false-negative

Example: learning a XOR b

Candidate solution





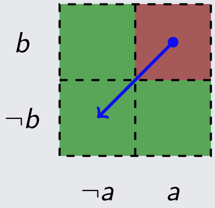
$$(\neg a \vee \neg b) \quad \wedge \quad (a)$$

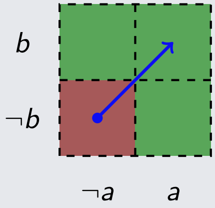
Operation

Check for false-negative $\rightarrow$ ✘: $\overline{a}b$

Example: learning a XOR b

Candidate solution





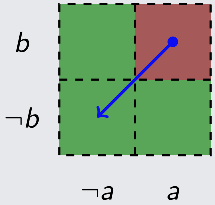
$(\neg a \vee \neg b) \quad \wedge \quad (a \vee b)$

Operation

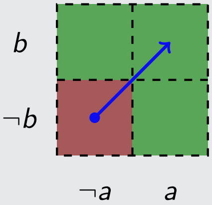
Check for false-positive

Example: learning a XOR b

Candidate solution



$(\neg a \vee \neg b)$ $\wedge$ $(a \vee b)$

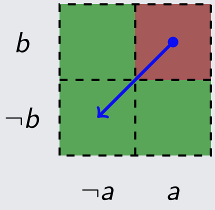


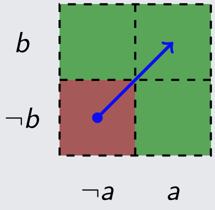
Operation

Check for false-positive $\rightarrow$ ✔

Example: learning a XOR b

Candidate solution





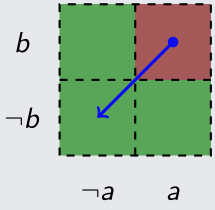
$(\neg a \vee \neg b) \quad \wedge \quad (a \vee b)$

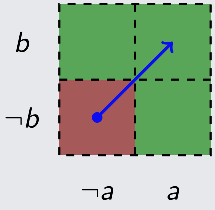
Operation

Check for false-negative

Example: learning a XOR b

Candidate solution





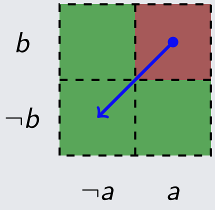
$(\neg a \vee \neg b) \quad \wedge \quad (a \vee b)$

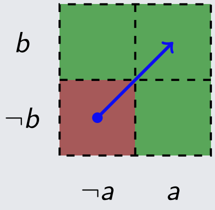
Operation

Check for false-negative $\rightarrow$ ✔

Example: learning a XOR b

Candidate solution





$$(\neg a \vee \neg b) \quad \wedge \quad (a \vee b)$$

Operation

No counter-examples? Done!

Experimental Setup

Benchmarks

- RATSY – GR(1) synthesis
  - IBM generalised buffer (Bloem et al., 2007b)
  - AMBA AHB Arbiter (Bloem et al., 2007a,b)
- UNBEAST – bounded synthesis
  - Load balancer (Ehlers, 2010)
  - Various benchmarks by Jobstmann and Bloem (2006)

Implementation

- Uses CUDD as BDD library
- Written in C++
- Imports a general strategy as CUDD BDD (written to a file)
- Postprocessing/verification: ABC and NuSMV (BMC)

Modes of operation

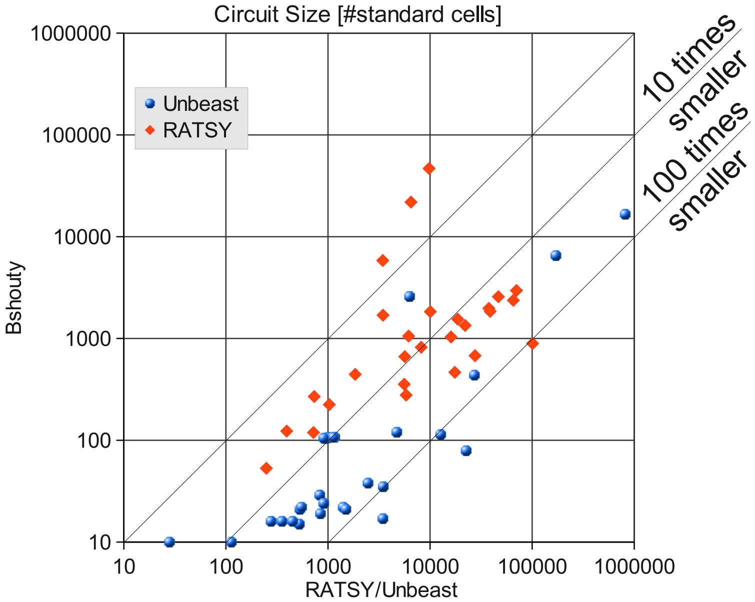
Any combination of ...

- ... DNF/CDNF leaning or their dual case, CNF/DCNF learning
- ... Smart variable order selection on/off

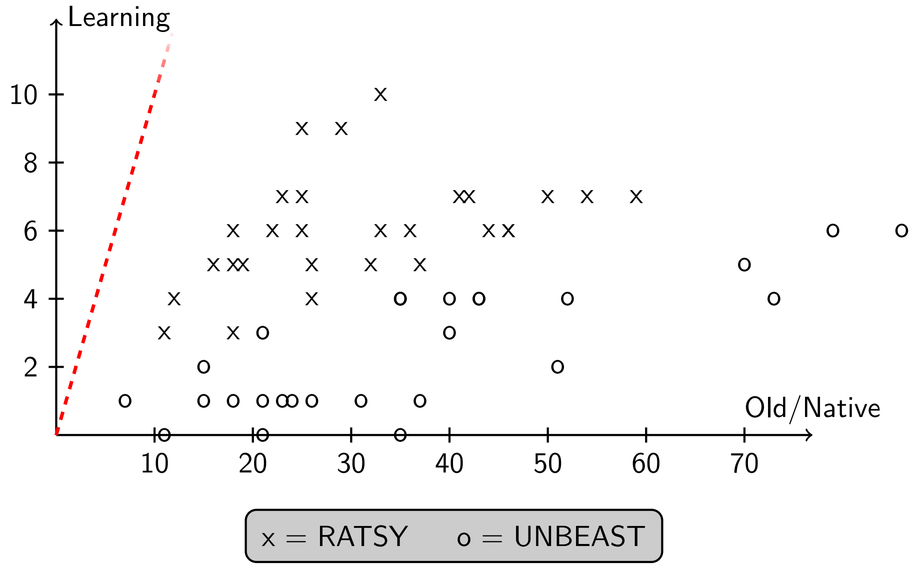
Best modes found

- **CDNF** learning + smart variable order selection **on**
- **DNF** learning + smart variable order selection **on**

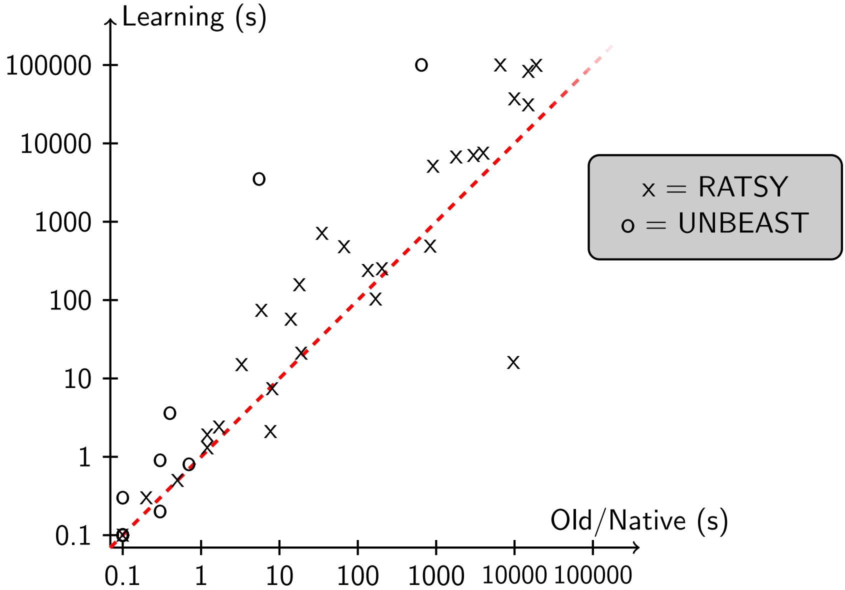
Circuit sizes (CDNF)



Circuit depths (CDNF)



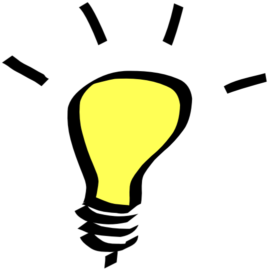
Computation times



Conclusion

Learning-based implementation extraction:

- an *efficient* and *effective* way to compute circuits in reactive synthesis
- *not restricted to BDDs*, but can be used with *any* symbolic data structure
- better suited for the task than all other techniques that we tried



Implementation available (for CuDD)

The URL is in the paper!

References I

D. Bañeres, J. Cortadella, and M. Kishinevsky. A recursive paradigm to solve Boolean relations. In *Design Automation Conference (DAC'04)*, pages 416–421. ACM, 2004. ISBN 1-58113-828-8.

R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *Design, Automation and Test in Europe (DATE'07)*, pages 1188–1193, 2007a.

R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electronic Notes in Theoretical Computer Science*, 190 (4):3–16, 2007b.

Rüdiger Ehlers. Symbolic bounded synthesis. In *CAV*, volume 6174 of *LNCS*, pages 365–379. Springer, 2010. ISBN 978-3-642-14294-9. doi: 10.1007/978-3-642-14295-6_33.

J. R. Jiang, H. Lin, and W. Hung. Interpolating functions from large Boolean relations. In *International Conference on Computer-Aided Design (ICCAD'09)*, pages 779–784. IEEE, 2009.

B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design (FMCAD'06)*, pages 117–124. IEEE, 2006. ISBN 0-7695-2707-8.

J. H. Kukula and T. R. Shiple. Building circuits from relations. In *Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 113–123. Springer, 2000. ISBN 3-540-67770-4.